CHAPTER 4
Unravelling Statistical Relationships

Introduction

Understanding the connection between different variables is part of unravelling statistical relationships. Covariance and correlation, outliers and probability distributions are critical to the unravelling of statistical relationships and make accurate interpretations based on data. Covariance and correlation essentially measure the same concept, the change in two variables with respect to each other. They aid in comprehending the relationship between two variables in a dataset and describe the extent to which two random variables or random variable sets are prone to deviate from their expected values in the same manner. Covariance illustrates the degree to which two random variables vary together. And correlation is a mathematical method for determining the degree of statistical dependence between two variables. Ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation). Statistical relationships are based on data and most data contains outliers. Outliers are observations that are significantly different from other data points, such as data variability or

experimental errors. Such outliers can significantly skew data analysis and statistical modeling, potentially leading to erroneous conclusions. Therefore, it is essential to identify and manage outliers to ensure accurate results. To facilitate comprehension and prediction of data patterns measuring likelihood and distribution of likelihood is required. For these statisticians use probability and probability distribution. The probability measures the likelihood of a specific event occurring and is denoted by a value between 0 and 1, where 0 implies impossibility and 1 signifies certainty.

A probability distribution which is a mathematical function describes how probabilities are spread out over the values of a random variable. For instance, in a fair roll of a six-sided dice, the probability distribution would indicate that each outcome (1, 2, 3, 4, 5, 6) has a probability of 1/6. While probability measures the likelihood of a single event, a probability distribution considers all potential events and their respective probabilities. It offers a comprehensive view of the randomness or variability of a particular data set. Sometimes there can be many data point or large data that need to be represented as one. In such case the data points in the form of arrays and matrices allow us to explore statistical relationships, distinguish true correlations from spurious ones, and visualize complex dependencies in data. All of these concepts in the structure below are basic, but very important steps in unraveling and understanding the statistical relationship.

Structure

In this chapter, we will discuss the following topics:

- Covariance and correlation
- Outliers and anomalies
- Probability

- Array and matrices

Objectives

By the end of this chapter, readers will see what covariance, correlation, outliers, anomalies are, how they affect data analysis, statistical modeling, and learning, how they can lead to misleading conclusions, and how to detect and deal with them. We will also look at probability concepts and the use of probability distributions to understand data, its distribution, and its properties, how they can help in making predictions, decisions, and estimating uncertainty.

Covariance

Covariance in statistics measures how much two variables change together. In other words, it is a statistical tool that shows us how much two numbers vary together. A positive covariance indicates that the two variables tend to increase or decrease together. Conversely, a negative covariance indicates that as one variable increases, the other tends to decrease and vice versa. Covariance and correlation are important in measuring association, as discussed in *Chapter 3, Frequency Distribution, Central Tendency, Variability.* While correlation is limited to -1 to +1, covariance can be practically any number. Now, let us consider a simple example.

Suppose you are a teacher with a class of students. And you observed when the temperature is high in the summer, the students' test scores generally decrease, while in the winter when it is low, the scores tend to rise. This is a negative covariance because as one variable, temperature, goes up, the other variable, test scores, goes down. Similarly, if students who study more hours tend to have higher test scores, this is a positive covariance. As study hours increase, test scores also increase. Covariance helps

identify the relationship between different variables.

Tutorial 4.1: An example to calculates the covariance between temperature and test scores, and between study hours and test scores, is as follows:

```
import numpy as np
# Let's assume these are the temperatures in Celsius
temperatures = np.array([30, 32, 28, 31, 33, 29, 34, 35, 36, 37])
# And these are the corresponding test scores
test_scores = np.array([70, 68, 72, 71, 67, 73, 66, 65, 64, 63])
# And these are the corresponding study hours
study_hours = np.array([5, 6, 7, 6, 5, 7, 4, 3, 2, 1])
# Calculate the covariance between temperature and test scores
cov_temp_scores = np.cov(temperatures, test_scores)[0, 1]
print(f"Covariance between temperature and test scores: {cov_temp_scores}")
# Calculate the covariance between study hours and test scores
cov_study_scores = np.cov(study_hours, test_scores)[0, 1]
print(f"Covariance between study hours and test scores: {cov_study_scores}")
```

Output:

```
Covariance between temperature and test scores: -10.277777777777777
Covariance between study hours and test scores: 6.733333333333334
```

As output shows, covariance between temperature and test score is negative (indicating that as temperature increases,

test scores decrease), and the covariance between study hours and test scores is positive (indicating that as study hours increase, test scores also increase).

Tutorial 4.2: Following is an example to calculates the covariance in a data frame, here we only compute covariance of selected three columns from the diabetes dataset:

```
# Import the pandas library and the display function
import pandas as pd
from IPython.display import display
# Load the diabetes dataset csv file
diabities_df = pd.read_csv("/workspaces/ImplementingStatisticsWithPython/data/chapter1/diabetes.csv")
diabities_df[['Glucose','Insulin','Outcome']].cov()
```

Output:

```
        Glucose     Insulin      Outcome
Glucose 1022.248314 1220.935799  7.115079
Insulin 1220.935799 13281.180078 7.175671
Outcome 7.115079    7.175671     0.227483
```

The diagonal elements (1022.24 for glucose, 13281.18 for insulin, and 0.22 for outcome) represent the variance of each variable. Looking at glucose its variance is 1022.24, which means that glucose levels vary quite a bit and insulin varies even more. Covariance between glucose and insulin is a positive number, which means that high glucose levels tend to be associated with high insulin levels and vice versa, and the covariance between insulin and outcome is 7.17. Since, these are positive numbers, this means that high glucose and insulin levels tend to be associated with high outcome and vice versa.

While covariance is a powerful tool for understanding relationships in numerical data, other techniques are typically more appropriate for text and image data. For

example, **term frequency-inverse document frequency (TF-IDF)**, cosine similarity, or word embeddings (such as **Word2Vec**) are often used to understand relationships and variations in text data. For image data, **convolutional neural networks (CNNs)**, image histograms, or feature extraction methods are used.

Correlation

Correlation in statistics measures the magnitude and direction of the connection between two or more variables. It is important to note that correlation does not imply causality between the variables. The correlation coefficient assigns a value to the relationship on a -1 to 1 scale. A **positive** correlation, closer to 1, indicates that as one variable increases, so does the other. Conversely, a **negative** correlation, closer to -1 means that as one variable increases, the other decreases. A correlation of zero suggests no association between two variables. More about correlation is also discussed in *Chapter 1, Introduction to Statistics and Data,* and *Chapter 3, Frequency Distribution, Central Tendency, Variability.* Remember that while covariance and correlation are related correlation provides a more interpretable measure of association, especially when comparing variables with different units of measurement.

Let us understand correlation with an example, consider relationship between study duration and exam grade. If students who spend more time studying tend to achieve higher grades, we can conclude that there is a positive correlation between study time and exam grades, as an increase in study time corresponds to an increase in exam grades. On the other hand, an analysis of the correlation between the amount of time devoted to watching television and test scores reveals a negative correlation. Specifically,

as the duration of television viewing (one variable) increases, the score on the exam (the other variable) drops. Bear in mind that correlation does not necessarily suggest causation. Mere correlation between two variables does not reveal a cause-and-effect relationship.

Tutorial 4.3: An example to calculates the correlation between study time and test scores, and between TV watching time and test scores, is as follows:

```
import numpy as np
# Let's assume these are the study hours
study_hours = np.array([5, 6, 7, 6, 5, 7, 4, 3, 2, 1])
# And these are the corresponding test scores
test_scores = np.array([70, 72, 75, 72, 70, 75, 68, 66, 64, 62])
# And these are the corresponding TV watching hours
tv_hours = np.array([1, 2, 1, 2, 3, 1, 4, 5, 6, 7])
# Calculate the correlation between study hours and test scores
corr_study_scores = np.corrcoef(study_hours, test_scores)[0, 1]
print(f"Correlation between study hours and test scores: {corr_study_scores}")
# Calculate the correlation between TV watching hours and test scores
corr_tv_scores = np.corrcoef(tv_hours, test_scores)[0, 1]
print(
    f"Correlation between TV watching hours and test scores: {corr_tv_scores}")
```

Output:

```
Correlation between study hours and test scores: 0.997128905932362 9
Correlation between TV watching hours and test scores: -0.9495412844036697
```

Output shows an increase in study hours correspond to a higher test score, indicating a positive correlation. A negative correlation is between the number of hours spent watching television and test scores. This suggests that an increase in TV viewing time is linked to a decline in test scores.

Outliers and anomalies

Outlier is a data point that significantly differs from other observations. It is a value that lies at an abnormal distance from other values in a random sample from a population. **Anomalies**, are similar to outliers as they are values in a data set that do not fit the expected behavior or pattern of the data. The terms outliers and anomalies are often used interchangeably in statistics, they can have slightly different connotations depending on the context. For example, let us say you are a teacher and you are looking at the test scores of your students. Most of the scores are between 70 and 90, but there is one score that is 150. This score would be considered an outlier because it is significantly higher than the rest of the scores. It is also an anomaly because it does not fit the expected pattern (since test scores usually range from 0 to 100). Another example is, in a dataset of human ages, a value of 150 would be an outlier because it is significantly higher than expected. However, if you have a sequence of credit card transactions and you suddenly see a series of very high-value transactions from a card that usually only has small transactions, that would be an anomaly. The individual transaction amounts might not be outliers by themselves, but the sequence or pattern of transactions is unusual given the past behavior of the card.

So, while all outliers could be considered anomalies (because they are different from the norm), not all anomalies are outliers (because they might not be extreme values, but rather unexpected pattern or behavior).

Tutorial 4.4: An example to calculates the concept of outliers and anomalies, is as follows:

```
1. import numpy as np
2. from scipy import stats
3. import matplotlib.pyplot as plt
4. # Let's assume these are the ages of a group of people
5. ages = np.array([20, 25, 30, 35, 40, 45, 50, 55, 60, 150]
   )
6. # Now let's consider a sequence of credit card transactio
   ns
7. transactions = np.
array([100, 120, 150, 110, 105, 102, 108, 2000, 2100, 220
   0])
8.
9. # Define a function to detect outliers using the Z-score
10. def detect_outliers(data):
11. outliers = []
12. threshold = 1
13. mean = np.mean(data)
14. std = np.std(data)
15. for i in data:
16. z_score = (i - mean) / std
17. if np.abs(z_score) > threshold:
18. outliers.append(i)
19. return outliers
Unravelling Statistical Relationships ν 141
20.
21. # Define a function to detect anomalies based on sudd
   en increase in transaction amounts
22. def detect_anomalies(data):
23. anomalies = []
```

```
threshold = 1.5 # this could be any value based on your understanding of the data
mean = np.mean(data)
for i in range(len(data)):
if i == 0:
continue # skip the first transaction
# if the current transaction is more than twice the previous one
if data[i] > threshold * data[i-1]:
anomalies.append(data[i])
return anomalies

anomalies = detect_anomalies(transactions)
print(f"Anomalies in transactions: {anomalies}")
outliers = detect_outliers(ages)
print(f"Outliers in ages: {outliers}")
# Plot ages with outliers in red
fig, (axs1, axs2) = plt.subplots(2, figsize=(15, 8))
axs1.plot(ages, 'bo')
axs1.plot([i for i, x in enumerate(ages) if x in outliers],
[x for x in ages if x in outliers], 'ro')
axs1.set_title('Ages with Outliers')
axs1.set_ylabel('Age')
# Plot transactions with anomalies in red
axs2.plot(transactions, 'bo')
axs2.plot([i for i, x in enumerate(transactions) if x in anomalies],
[x for x in transactions if x in anomalies], 'ro')
axs2.set_title('Transactions with Anomalies')
```

```
axs2.set_ylabel('Transaction Amount')
plt.savefig('outliers_anomalies.jpg', dpi=600, bbox_inches='tight')
plt.show()
```

In this program, we define two numpy arrays: **ages** and **transactions**, which represent the collected data. Two functions, **detect_outliers** and **detect_anomalies**, are then defined. The **detect_outliers** function uses the z-score method to identify outliers in the ages data. Likewise, the **detect_anomalies** function identifies anomalies in the transaction data based on a sudden increase in transaction amounts.

Output:

```
Anomalies in transactions: [2000]
Outliers in ages: [150]
```

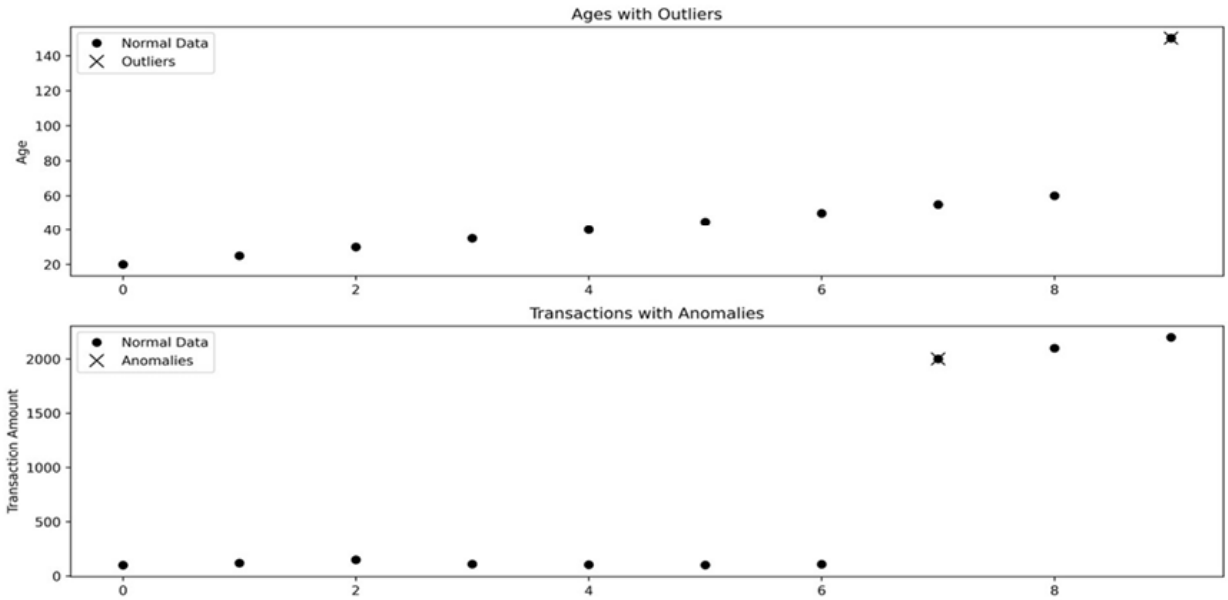

Figure 4.1: *Subplots showing outliers in age and anomalies in transaction*

The **detect_outliers** function identifies the age of 150 as an outlier, while the **detect_anomalies** function recognizes the transactions of 2000 as anomalies. marking a change in pattern with cross(x).

For textual data, an outlier could be a document or text entry that is considerably lengthier or shorter compared to the other entries in the dataset. An anomaly could occur when there is a sudden shift in the topic or sentiment of texts in a particular time series, or the use of uncommon words or phrases. For image data, an outlier could be an image that differs significantly in terms of its size, color distribution, or other measurable characteristics, contrasted with other images in the dataset. An anomaly is an image that includes objects or scenes that are not frequently found within the dataset. Detecting outliers and anomalies in image and text data often requires more intricate techniques compared to numerical data. These methods could involve **Natural Language Processing (NLP)** for text data and computer vision algorithms for image data. It is crucial to address outliers and anomalies correctly as they can greatly affect the efficiency of data analysis and machine learning models.

Tutorial 4.5: An example to demonstrates the concept of outliers in text data, is as follows:

```
import numpy as np
# Create a CountVectorizer instance to convert text data into a bag-of-words representation
from sklearn.feature_extraction.text import CountVectorizer
# Let's assume these are the text entries in our dataset
texts = [
    "I love to play football",
    "The weather is nice today",
    "Python is a powerful programming language",
    "Machine learning is a fascinating field",
    "I enjoy reading books",
    "The Eiffel Tower is in Paris",
    "Outliers are unusual data points that differ significan
```

```
    tly from other observations",
    "Anomaly detection is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data"
]
# Convert the texts to word count vectors
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
# Calculate the length of each text entry
lengths = np.array([len(text.split()) for text in texts])

# Define a function to detect outliers based on text length
def detect_outliers(data):
    outliers = []
    threshold = 1  # this could be any value based on your understanding of the data
    mean = np.mean(data)
    std = np.std(data)
    for i in data:
        z_score = (i - mean) / std
        if np.abs(z_score) > threshold:
            outliers.append(i)
    return outliers

outliers = detect_outliers(lengths)
print(
    f"Outlier text entries based on length: {[texts[i] for i, x in enumerate(lengths) if x in outliers]}")
```

Here, we first define a list of text entries. We then convert these texts to word count vectors using the **CountVectorizer** class from

sklearn.feature_extraction.text. This allows us to calculate the length of each text entry. We then define a function **detect_outliers** to detect outliers based on text length. This function uses the **z-score** method to detect outliers, similar to the method used for numerical data. The **detect_outliers** function should detect the last text entry as an outlier because it is significantly longer than the other text entries.

Output:

```
Outlier text entries based on length: ['Anomaly detection is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data']
```

In the output, the function **detect_outliers** is designed to identify texts that are significantly longer or shorter than the average length of texts in the dataset. The output text is considered an outlier because it contains more words than most of the other texts in the dataset.

For anomaly detection in text data, more advanced techniques are typically required, such as topic modeling or sentiment analysis. These techniques are beyond the scope of this simple example. Detecting anomalies in text data could involve identifying texts that are off-topic or have unusual sentiment compared to the rest of the dataset. This would require NLP techniques and is a large and complex field of study in itself.

Tutorial 4.6: An example to demonstrate detection of anomalies in text data, based on the **Z-score** method. Considering the length of words in a text, anomalies in this context would be words that are significantly longer than the average, is as follows:

```
import numpy as np

# Define a function to detect anamolies
```

```
def find_anomalies(text):
    # Split the text into words
    words = text.split()
    # Calculate the length of each word
    word_lengths = [len(word) for word in words]
    # Calculate the mean and standard deviation of the word lengths
    mean_length = np.mean(word_lengths)
    std_dev_length = np.std(word_lengths)
    # Define a list to hold anomalies
    anomalies = []
    # Find anomalies: words whose length is more than 1 standard deviations away from the mean
    for word in words:
        z_score = (len(word) - mean_length) / std_dev_length
        if np.abs(z_score) > 1:
            anomalies.append(word)
    return anomalies

text = "Despite having osteosarchaematosplanchnochondroneuromuelous and osseocarnisanguineoviscericartilaginonervomedullary conditions, he is fit."
print(find_anomalies(text))
```

Output:

```
['osteosarchaematosplanchnochondroneuromuelous', 'osseocarnisanguineoviscericartilaginonervomedullary']
```

Since, the words highlighted in the output have a z-score greater than one, they have been identified as anomalies. However, the definition of an outlier can change based on the context and the specific statistical methods you are using.

Probability

Probability is the likelihood of an event occurring, it is between 0 and 1, where 0 means the event is impossible and 1 means it is certain. For example, when you flip a coin, you can get either heads or tails. The chance of getting heads is 1/2 or 50%. That is because each outcome has an equal chance of occurring, and one of them is heads. Probability can also be used to determine the likelihood of more complicated events, such as the chance of getting two heads in a row is one in four, or 25%. For example, flipping a coin twice has four possible outcomes: heads-heads, heads-tails, tails-heads, tails-tails.

Probability consists of outcomes, events, sample space. Let us look at them in detail as follows:

- **Outcomes** are results of an experiment, like in coin toss head and tail are outcomes.
- **Events** are set of one or more outcomes and sample space is set of all possible outcomes. In the coin flip experiment, the event getting heads consists of the single outcome heads. In a dice roll, the event rolling a number less than 5 includes the outcomes 1, 2, 3, and 4.
- **Sample space** is set of all possible outcomes. For the coin flip experiment, the sample space is {heads, tails}. For the dice experiment, the sample space is {1, 2, 3, 4, 5, 6}.

Tutorial 4.7: An example to illustrate probability, outcomes, events, and sample space using the example of rolling dice, is as follows:

```
1. import random
2. # Define the sample space
3. sample_space = [1, 2, 3, 4, 5, 6]
4. print(f"Sample space: {sample_space}")
5. # Define an event
```

```
event = [2, 4, 6]
print(f"Event of rolling an even number: {sample_space}")
# Conduct the experiment (roll the die)
outcome = random.choice(sample_space)
# Check if the outcome is in the event
if outcome in event:
    print(f"Outcome {outcome} is in the event.")
else:
    print(f"Outcome {outcome} is not in the event.")
# Calculate the probability of the event
probability = len(event) / len(sample_space)
print(f"Probability of the event: {probability}.")
```

Output:

```
Sample space: [1, 2, 3, 4, 5, 6]
Event of rolling an even number: [1, 2, 3, 4, 5, 6]
Outcome 1 is not in the event.
Probability of the event: 0.5.
```

Probability distribution

Probability distribution is a mathematical function that provides the probabilities of occurrence of different possible outcomes in an experiment. Let us consider flipping a fair coin. The experiment has two possible outcomes, **Heads (H)** and **Tails (T).** Since the coin is fair, the likelihood of both outcomes is equal.

This experiment can be represented using a probability distribution, as follows:

- Probability of getting heads $P(H) = 0.5$
- Probability of getting tails $P(T) = 0.5$

In probability theory, the sum of all probabilities within a distribution must always equal 1, representing every

possible outcome of an experiment. For instance, in our coin flip example, $P(H) + P(T) = 0.5 + 0.5 = 1$. This is a fundamental rule in probability theory.

Probability distributions can be discrete and continuous as follows:

- **Discrete probability** distributions are used for scenarios with finite or countable outcomes. For example, you have a bag of 10 marbles, 5 of which are red and 5 of which are blue. If you randomly draw a marble from the bag, the possible outcomes are a red marble or a blue marble. Since there are only two possible outcomes, this is a discrete probability distribution. The probability of getting a red marble is 1/2, and the probability of getting a blue marble is 1/2.

Tutorial 4.8: To illustrate discrete probability distributions based on example of 10 marbles, 5 of which are red and 5 of which are blue, is as follows:

```
import random
# Define the sample space
sample_space = ['red', 'red', 'red', 'red', 'red', 'blue', 'blue', 'blue', 'blue', 'blue']
# Conduct the experiment (draw a marble from the bag)
outcome = random.choice(sample_space)
# Check if the outcome is red or blue
if outcome == 'red':
    print(f"Outcome is a: {outcome}")
elif outcome == 'blue':
    print(f"Outcome is a: {outcome}")
# Calculate the probability of the events
probability_red = sample_space.count('red') / len(sample_space)
probability_blue = sample_space.count('blue') / len(sample_space)
```

```
print(f"Overall probablity of drawing a red marble: {probability_red}")
print(f"Overall probablity of drawing a blue marble: {probability_blue}")
```

Output:

```
Outcome is a: red
Overall probablity of drawing a red marble: 0.5
Overall probablity of drawing a blue marble: 0.5
```

- **Continuous probability** distributions are used for scenarios with an infinite number of possible outcomes. For example, you have a scale that measures the weight of objects to the nearest gram. When you weigh an apple, the possible outcomes are any weight between 0 and 1000 grams. This is a continuous probability distribution because there are an infinite number of possible outcomes in the range of 0 to 1000 grams. The probability of getting any particular weight, such as 150 grams, is zero. However, we can calculate the probability of getting a weight within a certain range, such as between 100 and 200 grams.

Tutorial 4.9: To illustrate continuous probability distributions, is as follows:

```
import numpy as np
# Define the range of possible weights
min_weight = 0
max_weight = 1000
# Generate a random weight for the apple
apple_weight = np.random.uniform(min_weight, max_weight)
print(f"Weight of the apple is {apple_weight} grams")
# Define a weight range
min_range = 100
```

```
max_range = 200
# Check if the weight is within the range
if min_range <= apple_weight <= max_range:
    print(f"Weight of the apple is within the range of {min_range}-{max_range} grams")
else:
    print(f"Weight of the apple is not within the range of {min_range}-{max_range} grams")
# Calculate the probability of the weight being within the range
probability_range = (max_range - min_range) / (max_weight - min_weight)
print(f"Probability of the weight of the apple being within the range of {min_range}-
{max_range} grams is {probability_range}")
```

Output:

```
Weight of the apple is 348.2428034693577 grams
Weight of the apple is not within the range of 100-200 grams
Probability of the weight of the apple being within the range of 100-200 grams is 0.1
```

Uniform distribution

In uniform distribution, all possible outcomes are equally likely. The flipping a fair coin, is a uniform distribution. There are two possible outcomes: **Heads (H)** and **Tails (T).** Here, every outcome is equally likely.

Tutorial 4.10: An example to illustrate uniform probability distributions, is as follows:

```
import random
# Define the sample space
sample_space = ['H', 'T']
# Conduct the experiment (flip the coin)
```

```
outcome = random.choice(sample_space)
# Print the outcome
print(f"Outcome of the coin flip: {outcome}")
# Calculate the probability of the events
probability_H = sample_space.count('H') / len(sample_space)
probability_T = sample_space.count('T') / len(sample_space)
print(f"Probability of getting heads (P(H)): {probability_H}")
print(f"Probability of getting tails (P(T)): {probability_T}")
```

Output:

```
Outcome of the coin flip: T
Probability of getting heads (P(H)): 0.5
Probability of getting tails (P(T)): 0.5
```

Normal distribution

Normal distribution is symmetric about the mean, meaning that data near the mean is more likely to occur than data far from the mean. It is also known as the **Gaussian distribution** and describes data with bell-shaped curves. For example, measuring the test scores of 100 students. The resulting data would likely follow a normal distribution, with most students' scores falling around the mean and fewer students having very high or low scores.

Tutorial 4.11: An example to illustrate normal probability distributions, is as follows:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
# Define the parameters for the normal distribution,
# where loc is the mean and scale is the standard deviati
```

```
on.
# Let's assume the average test score is 70 and the standard deviation is 10.
loc, scale = 70, 10
# Generate a sample of test scores
test_scores = np.random.normal(loc, scale, 100)
# Create a histogram of the test scores
plt.hist(test_scores, bins=20, density=True, alpha=0.6, color='g')
# Plot the probablity distribution function
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, loc, scale)
plt.plot(x, p, 'k', linewidth=2)
title = "Fit results: mean = %.2f,  std = %.2f" % (loc, scale)
plt.title(title)
plt.savefig('normal_distribution.jpg', dpi=600, bbox_inches='tight')
plt.show()
```

Output:

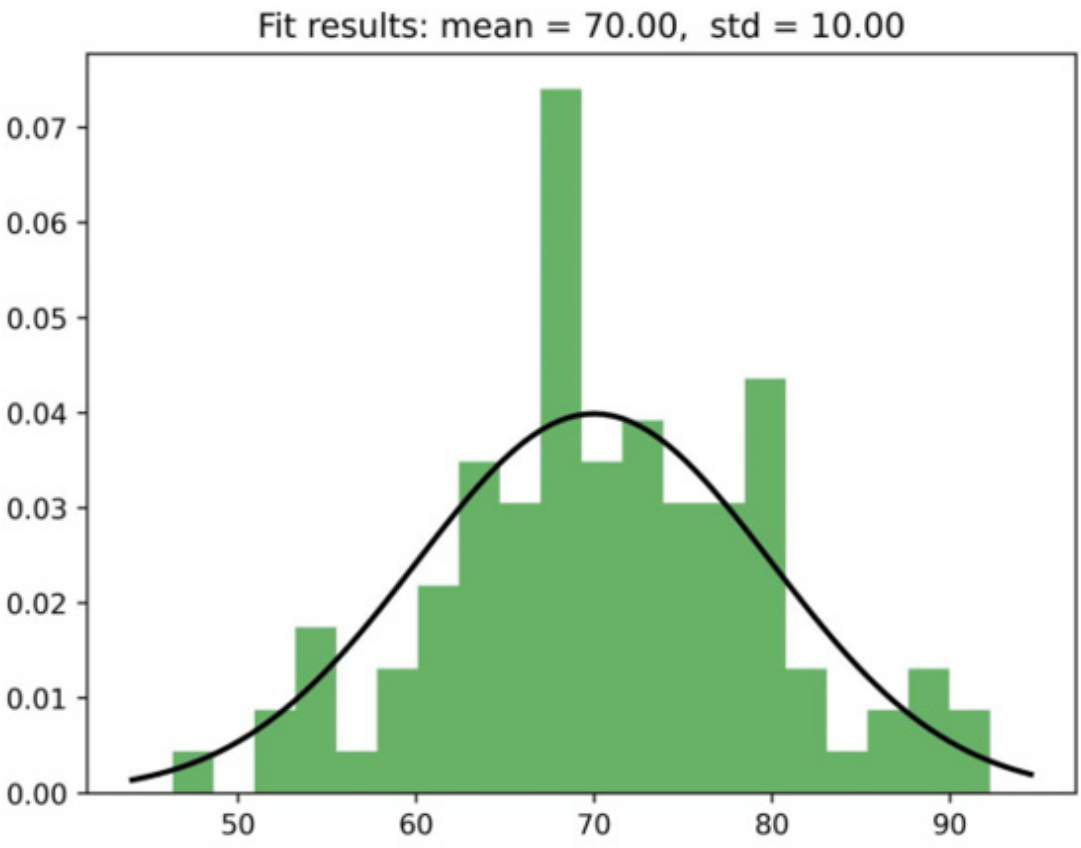

Figure 4.2: *Plot showing the normal distribution*

Binomial distribution

Binomial distribution describes the number of successes in a series of independent trials that only have two possible outcomes: success or failure. It is determined by two parameters, n, which is the number of trials, and p, which is the likelihood of success in each trial. For example, suppose you flip a coin ten times. There is a 50-50 chance of getting either heads or tails. For instance, the likelihood of getting strictly three heads is, we can use the binomial distribution to figure out how likely it is to get a specific number of heads in those ten flips.

For instance, the likelihood of getting strictly three heads, is as follows:

$$P(X = 3) = nCr * p^x * (1-p)^(n-x)$$

Where:

- nCr is the binomial coefficient, which is the number of ways to choose x successes out of n trials
- p is the probability of success on each trial (0.5 in this case)
- (1-p) is the probability of failure on each trial (0.5 in this case)
- x is the number of successes (3 in this case)
- n is the number of trials (10 in this case)

Substituting the values provided, we can calculate that there is a 12.16% chance of getting exactly 3 heads out of ten-coin tosses.

Tutorial 4.12: An example to illustrate binomial probability distributions, using coin toss example, is as follows:

```
from scipy.stats import binom
import matplotlib.pyplot as plt
import numpy as np
# number of trials, probability of each trial
n, p = 10, 0.5
# generate a range of numbers from 0 to n (number of tr
ials)
x = np.arange(0, n+1)
# calculate binomial distribution
binom_dist = binom.pmf(x, n, p)
# display probablity distribution of each
for i in x:
    print(
        f"Probability of getting exactly {i} heads in {n} flips i
s: {binom_dist[i]:.5f}")
# plot the binomial distribution
plt.bar(x, binom_dist)
plt.title(
```

```
    'Binomial Distribution PMF: 10 coin Flips, Odds of Success for Heads is p=0.5')
plt.xlabel('Number of Heads')
plt.ylabel('Probability')
plt.savefig('binomial_distribution.jpg', dpi=600, bbox_inches='tight')
plt.show()
```

Output:

```
Probability of getting exactly 0 heads in 10 flips is: 0.00098
Probability of getting exactly 1 heads in 10 flips is: 0.00977
Probability of getting exactly 2 heads in 10 flips is: 0.04395
Probability of getting exactly 3 heads in 10 flips is: 0.11719
Probability of getting exactly 4 heads in 10 flips is: 0.20508
Probability of getting exactly 5 heads in 10 flips is: 0.24609
Probability of getting exactly 6 heads in 10 flips is: 0.20508
Probability of getting exactly 7 heads in 10 flips is: 0.11719
Probability of getting exactly 8 heads in 10 flips is: 0.04395
Probability of getting exactly 9 heads in 10 flips is: 0.00977
Probability of getting exactly 10 heads in 10 flips is: 0.00098
```

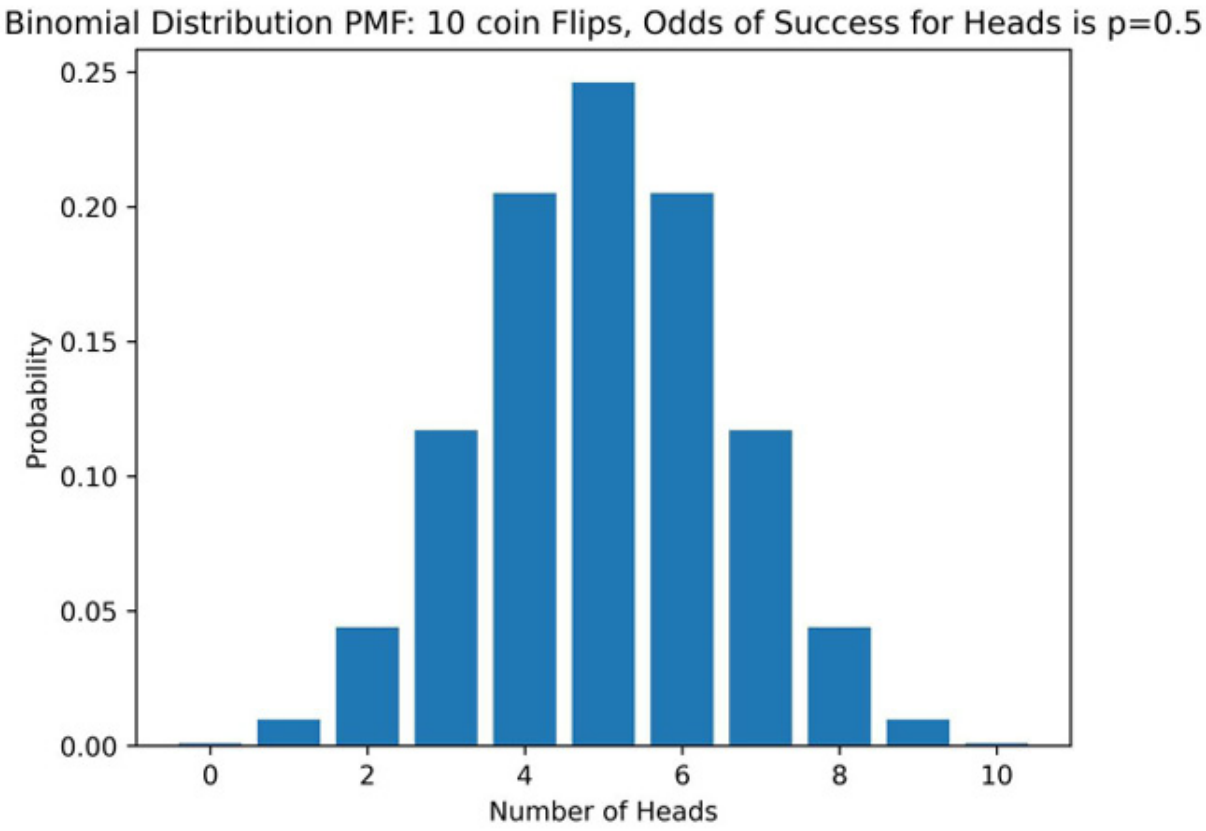

Figure 4.3: *Plot showing the normal distribution*

Poisson distribution

Poisson distribution is a discrete probability distribution that describes the number of events occurring in a fixed interval of time or space if these events occur independently and with a constant rate. The Poisson distribution has only one parameter, λ (lambda), which is the mean number of events. For example, assume you run a website that gets an average of 500 visitors per day. This is your λ (lambda). Now you want to find the probability of getting exactly 550 visitors in a day. This is a Poisson distribution problem because the number of visitors can be any non-negative integer, the visitors arrive independently, and you know the average number of visitors per day. Using the Poisson distribution formula, you can calculate the probability.

Tutorial 4.13: An example to illustrate Poisson probability

distributions, is as follows:

```
from scipy.stats import poisson
import matplotlib.pyplot as plt
import numpy as np
# average number of visitors per day
lambda_ = 500
# generate a range of numbers from 0 to 600
x = np.arange(0, 600)
# calculate Poisson distribution
poisson_dist = poisson.pmf(x, lambda_)
# number of visitors we are interested in
k = 550
prob_k = poisson.pmf(k, lambda_)
print(f"Probability of getting exactly {k} visitors in a day is: {prob_k:.5f}")
# plot the Poisson distribution
plt.bar(x, poisson_dist)
plt.title('Poisson Distribution PMF: λ=500')
plt.xlabel('Number of Visitors')
plt.ylabel('Probability')
plt.savefig('poisson_distribution.jpg', dpi=600, bbox_inches='tight')
plt.show()
```

We set **lambda_** to 500 in the program, representing the average daily visitors. The average number of visitors per day is 500. We generate numbers between 0 and 600 for x to cover your desired number of visitors, specifically 550. The program calculates and displays a bar chart of the Poisson distribution once executed. This chart represents the probability of receiving a specific number of visitors per day. The horizontal axis indicates the number of visitors, and the vertical axis displays the probability. The chart

displays the likelihood of having a certain number of visitors in a day. Each bar on the chart represents the probability of obtaining that exact number of visitors in one day.

Output:

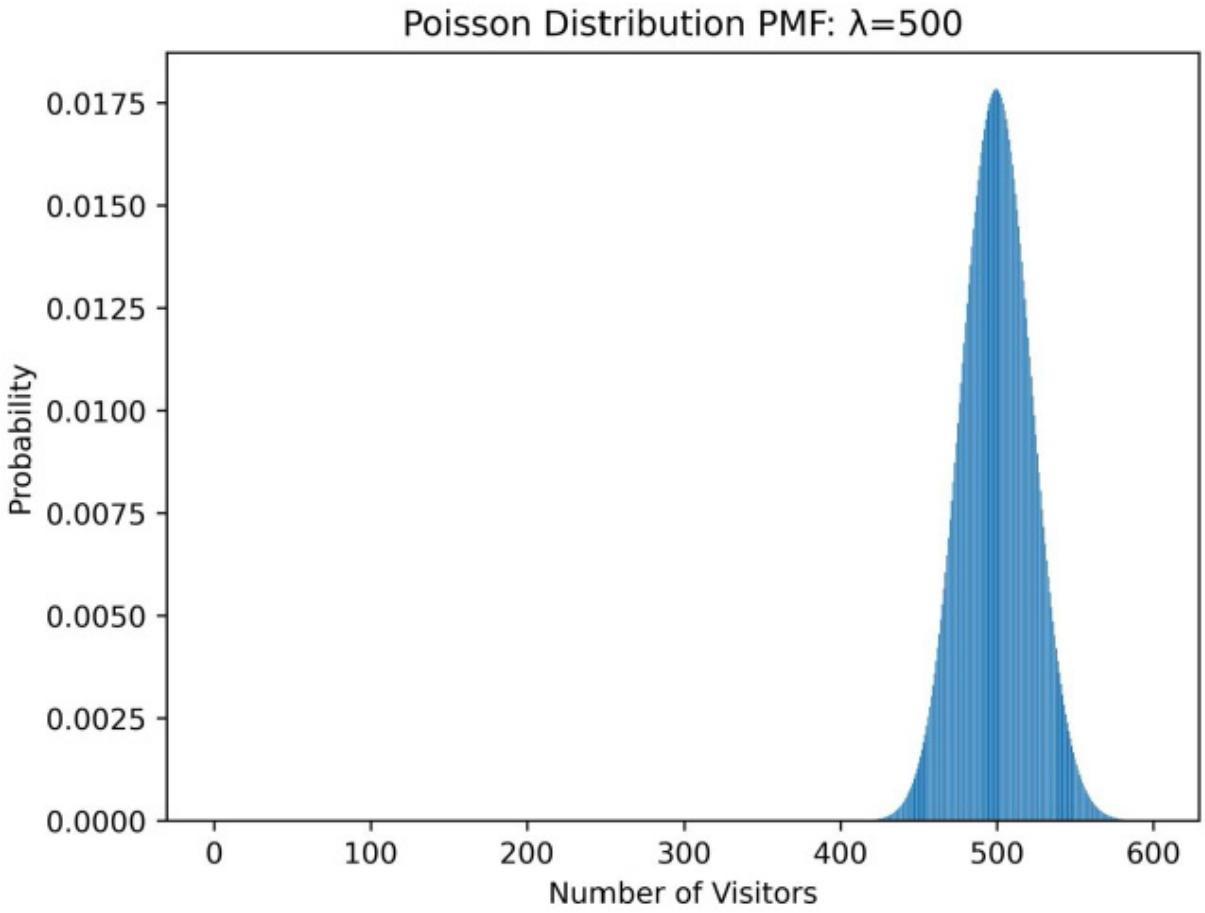

Figure 4.4: *Plot showing the Poisson distribution*

Array and matrices

Arrays are collections of elements of the same data type, arranged in a linear fashion. They are used to hold a collection of numerical data points, representing a variety of things, such as measurements taken over time, scores on a test, or other information. **Matrices** are 2-Dimensional arrays of numbers or symbols arranged in rows and

columns, used to organize and manipulate data in a structured way.

Arrays and matrices are fundamental structures to store and manipulate numerical data, crucial for statistical analysis and modeling. They provide a powerful and efficient way to store, manipulate, compute and analyze large datasets. Both array and matrices are used for the following:

- Storing and manipulating data
- Convenient and efficient mathematical calculation
- Statical modelling to analyze data and make prediction

Tutorial 4.14: An example to illustrate array or 1-Dimensional array, is as follows:

```
import statistics as stats
# Creating an array of data
data = [2, 8, 3, 6, 2, 4, 8, 9, 2, 5]
# Calculating the mean
mean = stats.mean(data)
print("Mean: ", mean)
# Calculating the median
median = stats.median(data)
print("Median: ", median)
# Calculating the mode
mode = stats.mode(data)
print("Mode: ", mode)
```

Output:

```
Mean:  4.9
Median:  4.5
Mode:  2
```

Tutorial 4.15: An example to illustrate 2-Dimensional array (which are matrix), is as follows:

```
import numpy as np
```

```
# Creating a 2D array (matrix) of data
data = np.array([[2, 8, 3], [6, 2, 4], [8, 9, 2], [5, 7, 1]])
# Calculating the mean of each row
mean = np.mean(data, axis=1)
print("Mean of each row: ", mean)
# Calculating the median of each row
median = np.median(data, axis=1)
print("Median of each row: ", median)
# Calculating the standard deviation of each row
std_dev = np.std(data, axis=1)
print("Standard deviation of each row: ", std_dev)
```

Output:

```
Mean of each row:  [4.33333333 4.         6.33333333 4.33333333]
Median of each row:  [3. 4. 8. 5.]
Standard deviation of each row:  [2.62466929 1.63299316 3.09120617 2.49443826]
```

Use of array and matrix

Use of array and matrices includes, using them to store large and wide data points and also be useful for analysis of those data. For example, use of matrix to storing data from surveys which consist of number of respondents in each age group or the average income for each education level. It can also be used in modeling of data. Let us look *Tutorial 4.16* and *Tutorial 4.17* to illustrate use of a matrix to store data from surveys which shows the number of respondents in each age group and the average income for each education level.

Tutorial 4.16: An example to illustrate use of a matrix to store data from surveys which shows the number of respondents in each age group, is as follows:

We first create a 2D array (matrix) to store the survey data.

With each row in the matrix stands for a survey taker, and every column corresponds to an attribute (like age range, education level, or earnings).

```
import numpy as np
# Creating a matrix to store survey data
data = np.array([
    ['18-24', 'High School', 30000],
    ['25-34', 'Bachelor', 50000],
    ['35-44', 'Master', 70000],
    ['18-24', 'Bachelor', 35000],
    ['25-34', 'High School', 45000],
    ['35-44', 'Master', 65000]
])
print("Data Matrix:")
print(data)
```

Output:

```
Data Matrix:
[['18-24' 'High School' '30000']
 ['25-34' 'Bachelor' '50000']
 ['35-44' 'Master' '70000']
 ['18-24' 'Bachelor' '35000']
 ['25-34' 'High School' '45000']
 ['35-44' 'Master' '65000']]
```

Tutorial 4.17: To extend above *Tutorial 4.16* for basic analysis of data matrix to compute the average income for each education level, is as follows:

```
import numpy as np
# Creating a matrix to store survey data
data = np.array([
    ['18-24', 'High School', 30000],
    ['25-34', 'Bachelor', 50000],
```

```
    ['35-44', 'Master', 70000],
    ['18-24', 'Bachelor', 35000],
    ['25-34', 'High School', 45000],
    ['35-44', 'Master', 65000]
])
# Calculating the number of respondents in each age group
age_groups = np.unique(data[:, 0], return_counts=True)
print("Number of respondents in each age group:")
for age_group, count in zip(age_groups[0], age_groups[1]):
    print(f"{age_group}: {count}")
# Calculating the average income for each education level
education_levels = np.unique(data[:, 1])
print("\nAverage income for each education level:")
for education_level in education_levels:
    income = data[data[:, 1] == education_level][:, 2].astype(np.float64)
    average_income = np.mean(income)
    print(f"{education_level}: {average_income}")
```

In this program, we first create a matrix to store the survey data. We then calculate the number of respondents in each age group by finding the unique age groups in the first column of the matrix and counting the occurrences of each. Next, we calculate the average income for each education level by iterating over the unique education levels in the second column of the matrix, filtering the matrix for each education level, and calculating the average of the income values in the third column.

Output:

```
Number of respondents in each age group:
18-24: 2
```

```
3. 25-34: 2
4. 35-44: 2
5.
6. Average income for each education level:
7. Bachelor: 42500.0
8. High School: 37500.0
9. Master: 67500.0
```

Conclusion

Understanding covariance and correlation is critical to determining relationships between variables, while understanding outliers and anomalies is essential to ensuring the accuracy of data analysis. The concept of probability and its distributions is the backbone of statistical prediction and inference. Finally, understanding arrays and matrices is fundamental to performing complex computations and manipulations in data analysis. These concepts are not only essential in statistics, but also have broad applications in fields as diverse as data science, machine learning, and artificial intelligence. Using covariance, correlation, observing outliers, anomalies, understanding of how data and probability concepts are used to predict outcomes and analyze the likelihood of events. All of these descriptive statistics concepts help to untangles statistical relationships. Finally, this covers descriptive statistics,

In *Chapter 5, Estimation and Confidence Intervals* we will start with the important concept of inferential statistics and how estimation is done, confidence interval is measured.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com